Advanced Graph Algorithms Jan-Apr 2014

Lecture 6 — January 15, 2014

Lecturer: Saket Sourabh *Scribe: Prafullkumar P Tale*

1 Overview

In the last lecture we defined *simple tree decomposition* and stated that for given graph G if (T, f) is *simple tree decomposition* then $|V(T)| \leq |V(G)|$.

In this lecture, we will prove the above result and introduce the concept of *nice tree decomposition* and how to convert *simple tree decomposition* to *nice tree decomposition* without increasing its tree width. We conclude this module with application of dynamic programming, using tree decomposition, to find the independent set of graph.

2 Simple Tree Decomposition

Definition 1. *For given graph G and its tree decomposition (T, f); T is said to be simple tree decomposition if for any two distinct nodes t_1 and t_2 in $V(T)$, neither $f(t_1) \not\subset f(t_2)$ nor $f(t_2) \not\subset f(t_1)$.*

Lemma 2. *For given graph G and its simple tree decomposition (T, f), $|V(T)| \leq |V(G)|$.*

Proof. Pick an arbitrary node $r \in V(T)$ and consider T as a rooted tree with r as the root. By the connectivity property of tree decomposition, for every vertex $v \in V(G)$ the set $\{b|v \in f(b)\}$ induces a connected subtree of T. Thus there is a unique node $b_v \in V(T)$ which is the node of T closest to r out of all nodes in $\{b|v \in f(b)\}$. We say that v *peaks* at b_v. Each vertex $v \in V(G)$ peaks at exactly one node b_v in $V(T)$.

We argue that for every node $b \in V(T)$ there is a vertex $v \in V(G)$ that peaks at b. Suppose not. If b is the root r this implies that $f(b) = \phi$, contradicting that (T, f) is simple as $f(b)$ will be contained in all $f(b_1)$ where $b_1 \in V(G)$ and $b \neq b_1$. If b is not the root r then b has a parent b' which is closer to the root r. Since no vertex $v \in V(G)$ peaks at b it follows that $f(b) \subseteq f(b')$ contradicting that (T, f) is simple.

We now have that every node b of $V(T)$ has some vertex G that peaks at it, and every vertex v of G peaks in exactly one node of T . Thus $|V(T)| \leq |V(G)|$. □

3 Nice Tree Decomposition

Definition 3. *A tree decomposition (T, f) of G is called nice if it has the following properties*

P1 *T is root at node r*

P2 *$f(r) = 0$ and for every leaf l, $f(l) = \phi$*

P3 *every node in b in T has at most 2 children*

P4 *If b has 2 children, say, b_1 and b_2 then $f(b) = f(b_1) = f(b_2)$.*

P5 *If b has only one child, say, b_1 then one of the following two conditions is true.*

(i) $f(b_1) \subset f(b)$ *and* $|f(b)| = |f(b_1)| + 1$

(ii) $f(b) \subset f(b_1)$ *and* $|f(b_1)| = |f(b)| + 1$

Consider the graph given in Fig 1 and its tree decomposition in Fig 2.

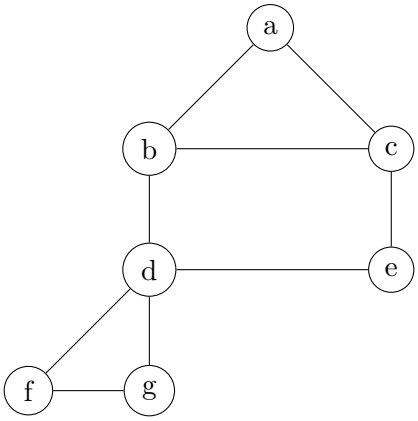


Figure 1: Graph G

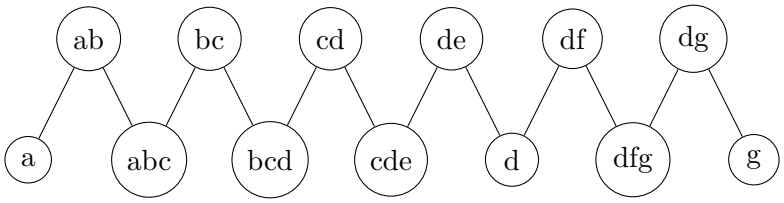


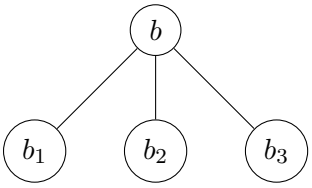
Figure 2: Nice Tree Decomposition of graph G

For node b, if the first case of property ***P5*** holds, then it is known as *introduce node*. If later case holds, we call it *forget node*.

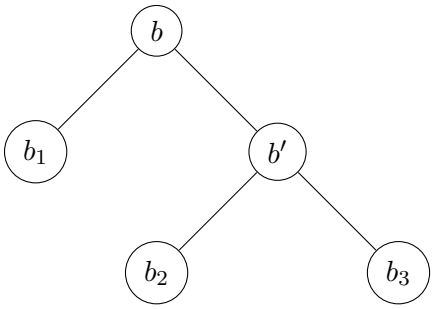
Lemma 4. *If graph G has tree-decomposition of tree-width k then it also has a nice tree decomposition of the same tree width. Furthermore, given any tree decomposition, we can construct nice tree decomposition in polynomial time.*

Proof. We start with considering T as an unrooted tree and setting $(T', f') = (T, f)$. We then proceed to modify (T', f') in order for it to have the desired properties. Throughout the process we will maintain that (T', f') is a tree decomposition of G of width k. First we add to every leaf l of T' another leaf l' and set $f'(l') = \phi$. Then we select an arbitrary leaf of T', call it r and root the tree T' at r. At this point (T', f') is a tree decomposition of G satisfying properties **P1** and **P2**.

As long as T' has a node $b \in V(T')$ with more than two children, we proceed as follows. Let $b_1, b_2 \dots b_p$ be the children of b. We make a new node b' and add it to T' in the following manner. The parent of b' is set to b, and for each child b_i of b with $i \geq 2$ we make b' the parent of b_i instead. Finally we set $f'(b') = f'(b)$. This operation maintains that (T', f') is a tree decomposition of G. Furthermore now b only have two children namely $b1$ and b'. The problem has shifted from b to b' but b' has degree one less than what b had before this step. Hence each step adds one node to the tree T' and for one vertex with more than two children, decreases the number of children by 1. We will continue this process till all the nodes in T' has at most two children. At this point (T', f') is a tree decomposition of G satisfying properties **P1**,**P2** and **P3**. Following is the simple illustration of the process when b has three children.

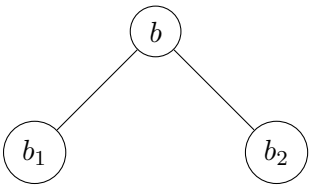


b with more than two children

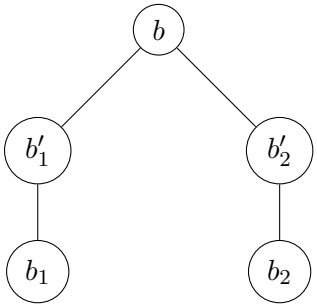


Introducing b' so that b now has two children

We now run a new process, aimed to make (T', f') satisfy property **P4**. As long as there exists a vertex $b \in V(T')$ with b_1, b_2 its children and $f'(b) \neq f'(b_1) \neq f'(b_2)$, we proceed as follows. We add two new nodes b'_1 and b'_2 to T'. Make b the parent of b'_1, b'_2 and change the parent of b_1, b_2 to be b'_1, b'_2 respectively. We set $f'(b'_1) = f'(b'_2) = f'(b)$. At the completion of step the number of vertices in $V(T')$ which contradicts to property **P4** is reduced by one. Hence, after finite iterations (T', f') is a tree decomposition of G satisfying properties **P1**,**P2**,**P3** and **P4**. Following diagram illustrate this step.



$f'(b) \neq' (b_1) \neq' (b_2)$



$f'(b) = f'(b'_1) = f'(b'_2)$

Finally we run a process aimed at making (T', f') satisfy property **P5**. Consider there exist vertex b in $V(T)$ with child b_1 such that $f(b) \subset f(b')$ and $|f(b)| + k = |f(b')|$ for some constant integer $k > 1$. We will introduce new vertex b' as child of b and make b' parent of b_1. Let $u \in f(b_1) \backslash f(b)$. Assign $f'(b') = f'(b) \cup u$. After this step, b satisfies the property **P5** but b' violate it. But, now

the difference between $f(b')$ and its child $f(b_1)$ is $k-1$. Hence after k iterations, all the newly introduced vertices in the path b to b_1 will satisfy the property **P5**. For vertex c in $V(T)$ with child c_1 such that $f(c) \supset f(c_1)$ and $|f(c)| = |f(c_1)| + k$. We introduce new vertex c' in similar way but we assign $f'(c') = f'(c) \backslash \{u\}$ where $u \in f'(c) \backslash f(c_1)$.

The total number of steps in all of the processes described above is $O(kn)$ since each step adds a new node to T'. With appropriate data structures, each step of each process can be completed in time $O(k^{O(1)})$ and hence we can construct nice tree decomposition in polynomial time. □

4 Dynamic Programming

Many algorithms on trees are based on dynamic programming and are polynomial time solvable. We defined tree width to measure how accurately a given graph can be approximated to tree. Lower the tree width of graph, more closely it can be associated with tree. Thus it is natural to think that some of the dynamic programming algorithms that work for trees could be lifted to also work on graphs of bounded treewidth. This turns out to be the case – with a few notable exceptions most problems that are polynomial time solvable on trees are also polynomial time solvable on graphs of bounded treewidth. We will use the independent set problem as an example to illustrate this concept.

Before jumping into the dynamic programming, we will first define some useful concepts. T_b is tree rooted at b and defined as $T_b = \{w | w \text{ is an descendant of } b \text{ in } T\} \cup \{b\}$.

Definition 5. *For given graph G and its tree decomposition (T, f), for $b \in V(T)$ define set $V_b = \bigcup_{w \in T_b} f(w)$*

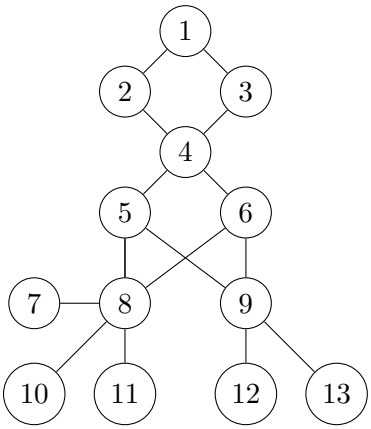
$V(G)$ is divided (note that it is not a partition) into V_b for all $b \in V(T)$. Define $V_1, V_2 \subseteq V(G)$ as $V_1 = V_b$ and $V_2 = \bigcup_{x \in V(T) \backslash T_b} f(x)$. Since G is t- decomposible, removing $t+1$ points will disconnect the graph and hence $V_1 \cap V_2 \leq t+1$.

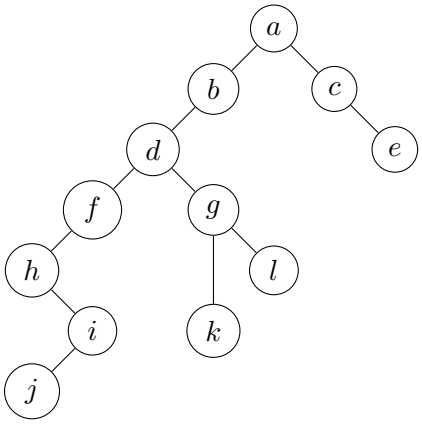
Definition 6. *For given graph G and $V_b \subseteq V(G)$, boundary of V_b is defined as $Z_{V_b} = \{v | v \in V_b, \exists (v, w) \in E(G)$ and $w \in V(G) \backslash V_b\}$*

Since graph is t decomposible, $|Z_{Vi}| \leq t+1$ for any $V_i \subseteq V(G)$. Let us consider the following example. We define function $\beta : V(G) \to 2^{V(T) \backslash \phi}$ and function $f : V(T) \to 2^{V(G)}$ as written in the table.

v	$\beta(v)$
1	$\{e\}$
2	$\{c, e\}$
3	$\{a\}$
4	$\{a, b\}$
5	$\{b, d, f\}$
6	$\{b, d, f, g\}$
7	$\{h\}$
8	$\{f, h, i, j\}$
9	$\{d, g, k, l\}$
10	$\{i\}$
11	$\{j\}$
12	$\{k\}$
13	$\{l\}$

x	$f(x)$
a	$\{4, 3\}$
b	$\{4, 5, 6\}$
c	$\{4, 2\}$
d	$\{5, 6, 9\}$
e	$\{2, 1\}$
f	$\{5, 6, 8\}$
g	$\{6, 9\}$
h	$\{7, 8\}$
i	$\{8, 10\}$
j	$\{8, 11\}$
k	$\{9, 12\}$
l	$\{9, 13\}$





In the above example, following are some of the subsets and their boundaries

- $V_h = \{7, 8, 10, 11\}$; $Z_{V_h} = \{8\}$
- $V_f = \{5, 6, 7, 8, 9, 10, 11\}$; $Z_{V_f} = \{5, 6\}$
- $V_d = \{5, 6, 7, 8, 10, 11, 9, 12, 13\}$; $Z_{V_d} = \{5, 6\}$

The idea is to store sufficient information in the boundary of the subset which will help us to determine Independent set of entire graph by looking only at boundary.

For $V_b \subseteq V(G)$ Let us define the following collection of sets

$$\mathcal{F}_b = \{A \cap V_{b_i} | A \text{ is independent set of } G \text{ and } V_{b_i} \subseteq V_b\}$$

Definition 7. *For given graph G its independent set A we define $A_1 = A \cap V_b$ and $A_2 = A \backslash A_1$. We say $\mathcal{F}_b$ is good family if there exist $B \in \mathcal{F}_b$ such that*

1. $B \cap Z_{V_b} = A_1 \cap Z_{V_b}$
2. $B \cup A_2$ *is an independent set.*
3. $|B \cup A_2| > |A|$.

But the size of family can be as large as $2^{|V(G)|}$. We can reduce the upper bound on the size of set by defining following collection of sets.

$$\hat{\mathcal{F}}_V(Z') = \{X | X \in F_V \text{ and } X \cap Z = Z'\}$$

where $Z' \subseteq Z$. This definition allow us select the better candidate among the V_{b_i} and leave out the candidate which forms the sub-optimal solution. We can reduce our focus to the above set. Since the size of boundary is bounded we now have 2^{t+1} possible subsets.

Now instead of working on graph G, we will work on its tree decomposition. As proved earlier, every graph has nice tree decomposition. We only need to see procedure at different type of nodes of nice tree viz leafs, nodes which has two children and introduce node or forget node.

We now describe the procedure and prove that the i^{th} family computed is good, by induction on i. First we compute the family $\mathcal{F}_b$ for every leaf b, and sets $\mathcal{F}_b = \{\phi\}$. Since (T, f) is a nice tree decomposition of G it follows that $f(b) = \phi$ and therefore $V_b = \phi$. Thus, for any optimal solution OPT of G, there is a set $B \in \mathcal{F}_b$, namely $B = \phi$, such that $B \cap f(b) = OPT \cap f(b) = \phi$ and $(OPT \backslash V_b) \cup B = OPT$ is an optimal solution to G.

Assume now that families of the i first nodes processed by the procedure are all good, and consider the $i + 1'$st node $b \in V(T)$ to be processed. If b is a leaf we have already shown that $\mathcal{F}_b$ is good, so assume now that b is not a leaf. All the children of b have been processed. The node b has one or two children in T, since (T, f) is a nice tree decomposition. If b has one child b_1 then $\mathcal{F}_{b_1}$ is good by the induction hypothesis. Similarly if b has two children b_1 and b_2 then $\mathcal{F}_{b_1}$ and $\mathcal{F}_{b_2}$ are both good by the induction hypothesis. It suffices to show that, in either case, the family $\mathcal{F}_b^*$ is good. If b has two children b_1 and b_2 then

$$\mathcal{F}_b^* = \{X_1 \cup X_2 | X_1 \in \mathcal{F}_{b_1}, X_2 \in \mathcal{F}_{b_2} \text{ and } X_1 \cap f(b) = X_2 \cap f(b)\}.$$

We wish to prove that $\mathcal{F}_b$ is good. To that end, consider any optimal solution $OPT \subseteq V(G)$. Since $\mathcal{F}_{b_1}$ is good there is a set $X_1 \in \mathcal{F}_{b_1}$ such that $OPT_1 = (OPT \backslash V_{b_1}) \cup X_1$ is an optimal solution as well, and $X_1 \cap f(b) = OPT \cap f(b)$. Here we used that (T, f) is a nice tree decomposition, and that therefore $f(b_1) = f(b_2) = f(b)$. Similarly, since $\mathcal{F}_{b_2}$ is good there is a set $X_2 \in \mathcal{F}_{b_2}$ such that $OPT_2 = (OPT_1 \backslash V_{b_1}) \cup X_2$ is an optimal solution, and $X_2 \cup f(b) = OPT_1 \cup f(b)$. Consider the set $X = X_1 \cup X_2$. The definition of $\mathcal{F}_b^*$ implies that $X \in \mathcal{F}_b^*$, and $OPT_2 = (OPT \backslash V(b)) \cup X$ is an optimal solution as well. Finally $X \cap f(b) = OPT \cap f(b)$. We conclude that in this case $\mathcal{F}_b^*$ is good.

If b has one child b_1 then

$$\mathcal{F}_b^* = \{X_1 \cup S | X_1 \in \mathcal{F}_{b_1} \text{ and } S \subseteq f(b) \backslash f(b_1)\}$$

We wish to prove that $\mathcal{F}_b^*$ is good. To that end, consider any optimal solution $OPT \subseteq V(G)$. Since $\mathcal{F}_{b_1}$ is good there is a set $X_1 \in \mathcal{F}_{b_1}$ such that $OPT_1 = (OPT \setminus V_{b_1}) \cup X_1$ is an optimal solution as well, and $X_1 \cap f(b_1) = OPT \cap f(b_1)$. Let $S = OPT \cap (f(b) \setminus f(b_1))$, and observe that S satisfies $S = OPT_1 \cap (f(b) \setminus f(b_1))$ as well. Let $X = X_1 \cup S$, we have that $X \in \mathcal{F}_{b_1}$, $OPT_1 = (OPT \setminus V_b) \cup X$ is an optimal solution, and $OPT \cap f(b) = S \cup (X_1 \cap f(b1) \cap f(b)) = X \cap f(b)$. Thus $\mathcal{F}_b^*$ is good family, concluding the proof of correctness of the procedure.